REAL-TIME IMAGE EDITING AND IOS APPLICATION WITH CONVOLUTIONAL NETWORKS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Ransen Niu

May 2018

ABSTRACT

This thesis presents a new image editing approach with convolutional networks to automatically alter the image content with a desired attribute and still keep the image photo-realistic. The proposed image editing approach effectively combines the strengths of two prominent images editing algorithms, conditional Generative Adversarial Networks[16] and Deep Feature Interpolation[19], to be time-efficient, memory-efficient, and user-controllable. We also present an inverted deep convolutional network to facilitate the proposed image editing approach. Lastly, we describe the implementation of this image editing approach in an iOS application and demonstrate that this approach is feasible and practical in real-world applications.

BIOGRAPHICAL SKETCH

Ransen Niu is graduating with an M.S. degree in the Department of Computer Science at Cornell University in May 2018. She works on Deep Learning and its applications in Computer Vision under the supervision of Prof. Kilian Weinberger. Currently, she researches on semantic image editing with convolutional networks. She received her bachelor degree from Purdue University in Computer Science in 2016. While at Purdue University, she researched on Machine Learning on information networks under the supervision of Prof. Jennifer Neville, with a focus on transfer learning across social networks.

ACKNOWLEDGEMENTS

I would like to thank Prof. Kilian Weinberger for his professional and valuable guidance. His support encourages me to finish this work in a good shape. I also want to thank Paul Upchurch for all the discussions we had along the way. And he has taught me a great deal about Computer Vision. I'm also grateful to everyone in my lab. They are always willing to help me with problems I encounter in both research and life. Finally, I would also like to extend my thanks to my parents and friends. They are always there for me no matter what.

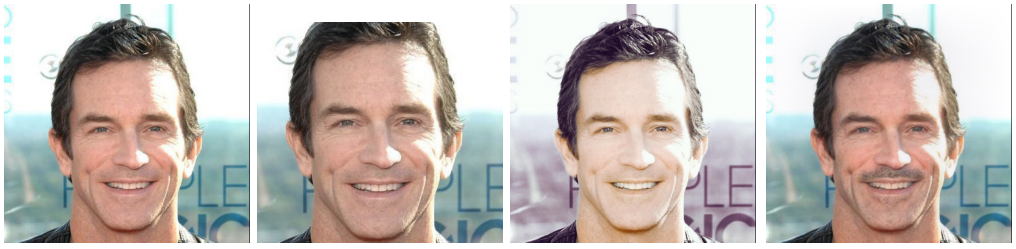
TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Recent years have witnessed a surge in the popularity of automatic photo editing applications, such as Snapseed, which has more than 50 million downloads on the Google Playstore. This surge has been largely fueled by the increasing number of digital photos people are taking thanks to the sophistication of mobile phone cameras. In the past, people have used professional photo editing softwares like Adobe Photoshop, but those require basic knowledge of graphic design, making it hard for non-professionals to use. In contrast, automatic photo editing applications simplify the image editing process and serve the need of the general population. People can be as artistic as they would like using automatic photo editing applications.

Figure 1 demonstrates several image editing tasks. While cropping or applying filters to an image is a basic photo processing technique, adding mustache to a face is more challenging. Adding mustache is an example of semantic image editing, where the content of an image is altered but the altered image is



Original Cropped With filtering Adding mustache

Figure 1: Given the original image, the right three images are examples of possible image editing. Adding mustache uses our image editing approach.

Figure 2: Images are edited to add mustache with an increasing intensity from left to right using our approach.

still photo-realistic. There have been a great number of works on semantic image editing[9][10][18][19][21]. [10][18] edit images using traditional computer vision algorithms, such as image warping and image composition. [18] opens closed eyes by finding an appropriate set of eyes in the user's personal photo collection, and transfers them onto a target face image. Recently, convolutional networks have been extensively used for image editing. Conditional Generative Adversarial Networks[16] have achieved outstanding performance on image editing[9][21]. Deep Feature Interpolation[19] introduces a data-driven image editing algorithm that relies on deep convolutional neural networks.

In this thesis, we present a new real-time semantic image editing approach with convolutional networks. We also implement this image editing approach in an iOS application to demonstrate that our approach is practical for real-world applications. We design this approach with three characteristic goals: **Time-efficiency.** To ensure that images can be edited with this approach in real time. **Memory-efficiency.** To ensure that the memory usage is minimal so that it can be implemented on mobile devices. **User-controllability.** To allow users to control the intensity of the image editing. For example, if a user wants to add mustache to a face in an image, the user is able to control how much mustache is to be added to the face (Figure 2). Therefore, other than the input image to be edited, our approach also takes in a control factor to regulate the intensity of the

image editing applied to the input image.

Our image editing approach achieves the three characteristic goals by effectively combining two prominent image editing algorithms, conditional Generative Adversarial Networks (cGANs)[16] and Deep Feature Interpolation (DFI)[19]. We use cGANs to translate an input image directly to an output image with a desired attribute. Then we adopt the idea of linear interpolation from DFI to adjust the intensity of the desired attribute in the output image in deep feature space in accordance with the control factor. Lastly, we map the output image from deep feature space back to pixel space. To the best of our knowledge, this thesis is the first work combining cGANs and DFI for a time-efficient, memory-efficient, and user-controllable image editing approach.

In this thesis, we are going to demonstrate our image editing approach and other approaches by editing facial images, but our approach can still be adapted to other kinds of image editing. Chapter 2 gives background on cGANs and DFI, and describe the strengths and weaknesses of each algorithm. Chapter 3 describes our image editing approach in detail, including how to effectively combine cGANs and DFI. Chapter 4 shows the implementation of our image editing approach in an iOS application. Chapter 5 summarizes the contribution of this thesis and discusses the future work.

CHAPTER 2

BACKGROUND ON IMAGE EDITING ALGORITHMS

2.1 Conditional Generative Adversarial Networks

Currently, one of the most prominent approaches for image editing is cGANs. cGANs are an extension of Generative Adversarial Networks (GANs)[6]. GANs consist of two networks, a generator G and a discriminator D. The generator and the discriminator are trained simultaneously and compete against each other in a two-player minimax game. G learns to generate images that cannot be distinguished from real images by an adversarially trained D. And D treats the generated images as negative examples to learn a better discrimination between real images and generated images.

Formally, the generator G learns to map a prior distribution $P_z(\mathbf{z})$ to a probability distribution $P_G(\mathbf{y})$ over images $\mathbf{y}$. Given a random noise vector $\mathbf{z}$, G outputs an image $\mathbf{y}'$ in the distribution $P_G(\mathbf{y})$. And the discriminator D learns to a single scalar in [0,1], representing the probability of an image coming from the

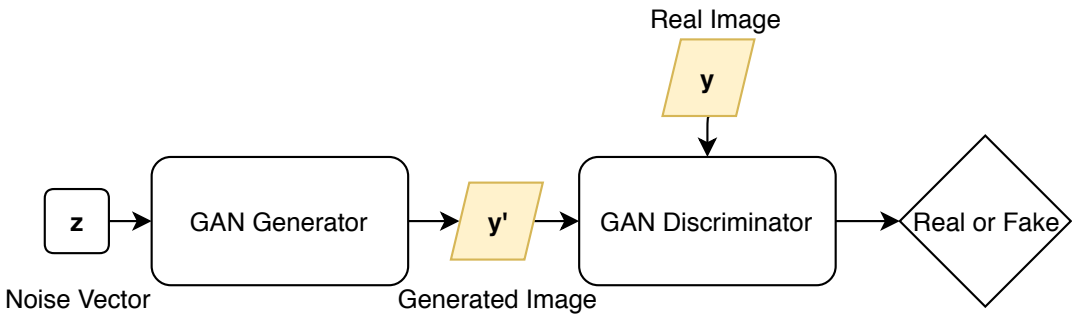


Figure 3: The structure of GANs.

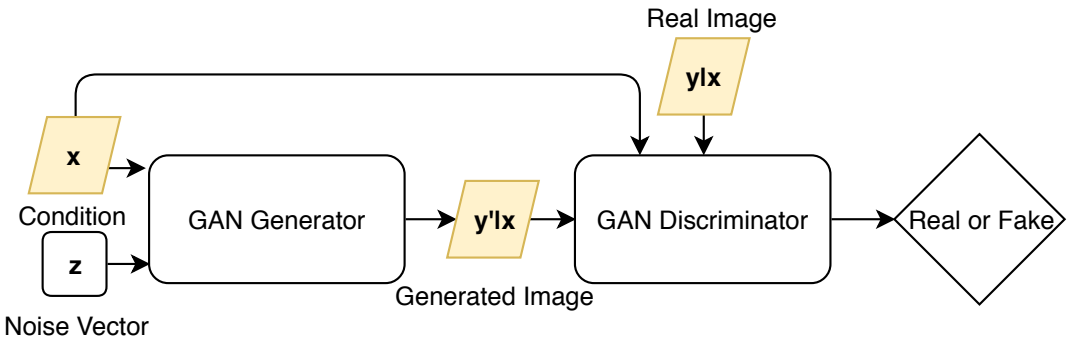


Figure 4: The structure of cGANs.

generated probability distribution $P_G(\mathbf{y})$ (i.e., close to 0) or the real probability distribution $P_{data}(\mathbf{y})$ over images $\mathbf{y}$ (i.e., close to 1). The structure of GANs is illustrated as Figure 3.

The objective equation for GANs is

$$G = \arg\min_G \max_D \mathcal{L}_{GAN}(G, D) \tag{2.1}$$

where G tries to minimize the loss L_{GAN} against an adversarial D which tries to maximize L_{GAN}, and the loss function L_{GAN} is expressed as:

$$\mathcal{L}_{GAN}(G, D) = \mathbf{E}_{\mathbf{y}\sim p_{data}(\mathbf{y})}[logD(\mathbf{y})] + \mathbf{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \tag{2.2}$$

While there is no control on what kinds of images are generated with an unconditioned generator G in GANs, cGANs[16] extend GANs to a conditional model on some extra information $\mathbf{x}$ to direct the generation process. Both the generator G and the discriminator D in cGANs are taking $\mathbf{x}$ as an additional input. The structure of cGANs is illustrated as Figure 4. And the loss function L_{cGAN} is:

$$\mathcal{L}_{CGAN}(G, D) = \mathbf{E}_{\mathbf{y}\sim p_{data}(\mathbf{y})}[logD(\mathbf{y}|\mathbf{x})] + \mathbf{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z}|\mathbf{x})|\mathbf{x}))] \tag{2.3}$$

In an image editing task, cGANs are conditioned on the image to be edited. A lot of works have further investigated image editing tasks using cGANs[9][21][11]. *Pix2pix*[9] models a discriminator to penalize structures at the scale of fix-sized patches. However, cGANs require paired data of an input image and an expected output image for training. The reason is that G now outputs a generated image conditioned on $\mathbf{x}$, which is the negative example to train D, so the positive example to train D should be the ground-truth image with expected editing correspondingly. CycleGAN[21] and DiscoGAN[11] use a cycle consistency technique makes it possible to train cGANs without paired data. They impose transitivity by forcing the transformation in one direction followed by the transformation in the opposite direction to lead back to the original input image.

In general, the image editing process only needs a single forward pass through the generator G, so cGANs are usually time-efficient and memory-efficient for image editing. However, cGANs doesn't allow users to control the intensity of image editing since the model is trained with images with a fixed intensity of the desired attribute.

2.2 Deep Feature Interpolation

While cGANs have been extensively deployed in image editing tasks, Deep Feature Interpolation (DFI)[19] stands out as a novel data-driven image editing algorithm that relies on simple linear interpolation in deep feature space. DFI is based on the hypothesis [2] that deep convolutional neural networks can linearize the image manifold by mapping images from pixel space into deep fea-

ture space. DFI can be described in the following four steps.

Step 1: Given an input image $\mathbf{x}$ and a desired attribute, DFI searches in a face dataset for k nearest neighbor images of $\mathbf{x}$ as a source image set $\mathbf{S}$, where each image $s_i \in \mathbf{S}$ shares similar attributes as $\mathbf{x}$ but without the desired attribute. DFI also searches for a target image sets $\mathbf{T}$, where each image $t_i \in \mathbf{T}$ shares similar attributes as $\mathbf{x}$ and with the desired attribute. These neighbors are selected by counting how many attributes they share with $\mathbf{x}$. Pre-trained classifiers are used to determine if an image has a certain attribute or not.

Step 2: Images in the source set $\mathbf{S}$ and the target set $\mathbf{T}$ are mapped to deep feature representations in deep feature space through a pre-trained deep convolutional neural network ϕ. DFI computes the mean feature representations for each image set and defines their difference as the desired attribute feature representation, denoted as $\mathbf{w}$:

$$\mathbf{w} = \frac{1}{|\mathbf{T}|} \sum_i \phi(t_i) - \frac{1}{|\mathbf{S}|} \sum_i \phi(s_i) \tag{2.4}$$

Step 3: DFI obtains the deep feature representation $\phi(\mathbf{x})$ of the input image $\mathbf{x}$ through ϕ. With a parameter α to control the magnitude of $\mathbf{w}$, the deep feature presentation $\phi(\mathbf{y})$ for an output image $\mathbf{y}$ can be constructed with linear interpolation:

$$\phi(\mathbf{y}) = \phi(\mathbf{x}) + \alpha\mathbf{w} \tag{2.5}$$

Step 4: Given the output image feature representation $\phi(\mathbf{y})$, DFI uses gradient descent[15] on the deep convolutional network to seek for the output image $\mathbf{y}$

in pixel space:

$$\mathbf{y} = \arg\min_{\mathbf{y}} \|(\phi(\mathbf{x}) + \alpha\mathbf{w}) - \phi(\mathbf{y})\|_2^2 + \lambda R_V(\mathbf{y}) \tag{2.6}$$

where R_V is the total variation regularizer to encourage smooth transitions between pixels.

To summarize, DFI performs image editing with simple linear interpolation of deep feature representations from a pre-trained deep convolutional network. DFI is able to control the intensity of the image editing by adjusting the magnitude of the attribute feature representation $\mathbf{w}$. However, DFI needs to search for k nearest neighbors in a huge face dataset, which makes DFI memory-inefficient. And searching for the k nearest neighbors is time-consuming since every image in the face dataset needs to be evaluate for similarity. Moreover, **Step 4** is also time-inefficient because it requires computing gradients of deep feature representations during inference time.

CHAPTER 3

REAL-TIME IMAGE EDITING

As mentioned in Chapter 2, cGANs and DFI are two prominent image editing algorithms, but each of them only shares some of the design goals with our image editing approach. While cGANs are time-efficient and memory-efficient, they can't control the intensity of the image editing. On the other hand, DFI can control the intensity of the image editing, but it's time-inefficient and it requires large memory space for the face dataset.

Therefore, we combine the strengths of cGANs and DFI, together with deep convolutional networks, to develop a time-efficient, memory-efficient, and user-controllable image editing approach. Our image editing approach uses cGANs to compose the desirable attribute for a specific image in a fast and light-weighted way. It also deploys the linear interpolation approach in DFI to allow controls of the image editing. Deep convolutional networks are used for mappings between pixel space and deep feature space.

3.1 Approach

Given an input image $\mathbf{x}$, we want to output an image $\mathbf{y}_\alpha$ by editing $\mathbf{x}$ with a desired attribute of an intensity α. As illustrated in Figure 5, the input image $\mathbf{x}$ first goes through the cGAN generator G to output an intermediate image $\mathbf{x'}$. Then both the input and the intermediate images obtain their deep feature representations, $\phi(\mathbf{x})$ and $\phi(\mathbf{x'})$ respectively, through a deep convolutional network ϕ. Next, the desired attribute feature representation $\mathbf{w}$ and the output image

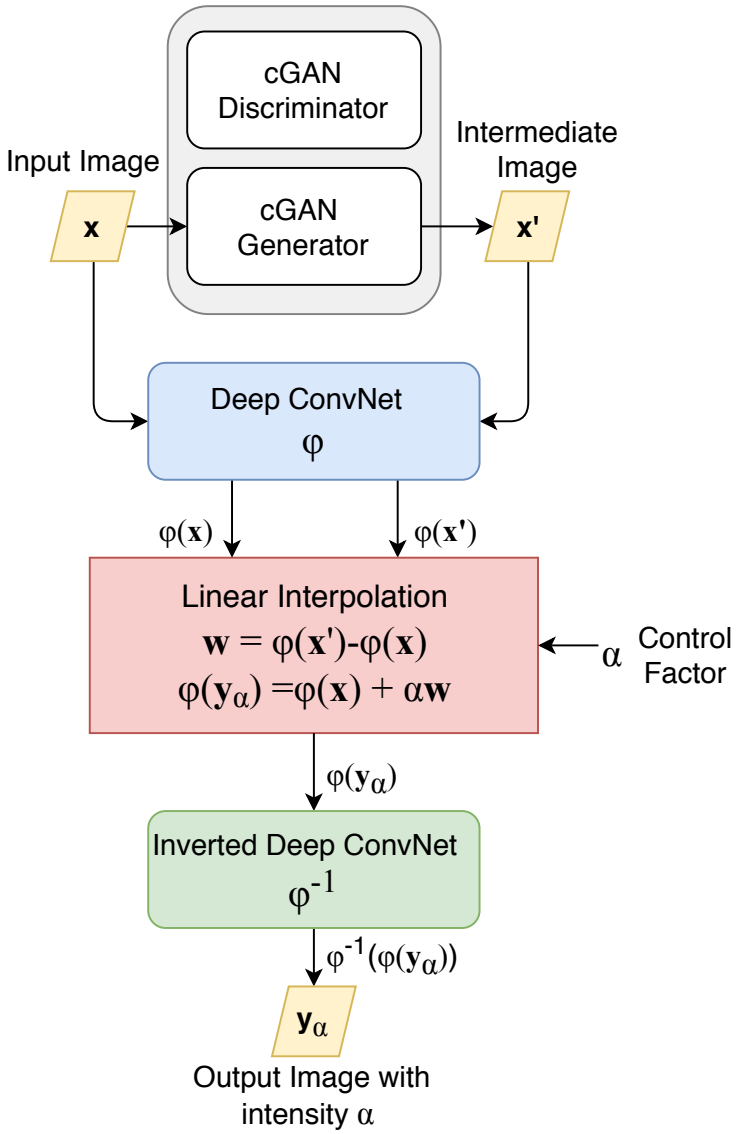


Figure 5: The overall architecture of our image editing approach.

feature representation $\phi(\mathbf{y}_\alpha)$ are constructed consecutively. The control factor α constrains the intensity of the attribute to be added. Lastly, the output image feature representation $\phi(\mathbf{y}_\alpha)$ is mapped to the output image $\mathbf{y}_\alpha$ in pixel space through an inverted deep convolutional network ϕ^{-1}.

3.1.1 Composition of the Desired Attribute

In order to control the intensity of the desired attribute, we need to compose the desired attribute for the input image $\mathbf{x}$. However, composing the desired attribute is difficult since the attribute should be customized to the input image. This is how we ensure that the output image is photo-realistic. For example, if we are adding mustache, we would expect grey mustache for people with grey hair, and black mustache for people with black hair. DFI composes the desired attribute with the mean difference between a source image set where every image looks like the input image without the desired attribute and a target image set where every image looks like the input image but with the desired attribute. However, as mentioned in Chapter 2, this step in DFI is neither time-efficient nor memory-efficient.

Therefore, we use cGANs, which are fast and light-weighted, to compose the desired attribute. We train a specific cGAN model such that it takes the input image $\mathbf{x}$ and output an image $\mathbf{x'}$ with the desired attribute. We can define the difference between the image $\mathbf{x'}$ and the input image $\mathbf{x}$ as the desired attribute in pixel space. We refer the image $\mathbf{x'}$ as the intermediate image since its role is to compose the desired attribute rather than being the final output image.

3.1.2 Linear Interpolation in Deep Feature Space

cGANs make it efficient to compose the desired attribute for a specific input image $\mathbf{x}$. However, images in pixel space lie on image manifolds that are locally euclidean but globally non-euclidean. It means that we can't directly apply linear interpolation on the desired attribute in pixel space. Hence, we adopt the idea from DFI that deep convolutional networks can linearize image manifolds, and perform linear interpolation in deep feature space.

Instead of directly composing the desired attribute in pixel space as described in Section 3.1.1, we compose the deep feature representation of the desired attribute. We first map the input image $\mathbf{x}$ and the intermediate image $\mathbf{x'}$ from cGANs to their deep feature representations, $\phi(\mathbf{x})$ and $\phi(\mathbf{x'})$ respectively, through a deep convolutional network ϕ. And we define the difference between the two deep feature representations as the deep feature representation $\mathbf{w}$ of the desired attribute:

$$\mathbf{w} = \phi(\mathbf{x'}) - \phi(\mathbf{x}) \tag{3.1}$$

then we use linear interpolation to get the deep feature representation for the final output image $\phi(\mathbf{y}_\alpha)$:

$$\phi(\mathbf{y}_\alpha) = \phi(\mathbf{x}) + \alpha\mathbf{w} \tag{3.2}$$

where α is the control factor to adjust the magnitude of the desired attribute in the final output image.

3.1.3 Inverted Deep Convolutional Networks

With linear interpolation in deep feature space, we are able to get the deep feature representation for the final output image $\phi(\mathbf{y}_\alpha)$ with a user-selected control factor α. In this step, we want to map $\phi(\mathbf{y}_\alpha)$ into pixel space to retrieve the output image $\mathbf{y}_\alpha$. DFI obtains $\mathbf{y}_\alpha$ with gradient descent on the deep convolutional neural network by optimizing the objective function (Equation 2.6), which requires computing gradients during inference time. Moreover, this objective function (Equation 2.6) only minimizes the difference between deep feature representations without taking the image reconstruction error into account.

There have been several publications on reverse mapping[3][5], where they train an inverted network to reverse the input and the output of the original network. [5] replaces convolutional layers with deconvolutional layers, layer by layer, to construct an inverted network for AlexNet[13]. They optimize the squared Euclidean reconstruction error between an image $\mathbf{y}$ and a reconstructed image $\phi^{-1}_{Alex}(\phi_{Alex}(\mathbf{y}))$:

$$\mathcal{L}_{Alex} = \|(\mathbf{y} - \phi^{-1}_{Alex}(\phi_{Alex}(\mathbf{y})))\|^2_2 \tag{3.3}$$

where $\mathbf{y}$ is the ground-truth image to be generated, ϕ_{Alex} is AlexNet, and ϕ^{-1}_{Alex} is the inverted AlexNet in their paper.

In our image editing approach, we propose to train an inverted deep convolutional neural network ϕ^{-1} of the deep convolutional network ϕ. This inverted network takes in a deep feature representation $\phi(\mathbf{y})$ as the input and outputs a corresponding image $\hat{\mathbf{y}}$ through a single forward pass. We design the architecture of ϕ^{-1} in the same way as [5]: each convolutional layer in ϕ is replaced with a deconvolutional layer in ϕ^{-1}. However, in the training process, we combine

DFI's reverse mapping (Equation 2.6) and [5]'s loss function (Equation 3.3) to construct a more well-defined loss function, taking in account both the perceptual loss and the image reconstruction error:

$$\mathcal{L}_{inverted} = \|(\mathbf{y} - \hat{\mathbf{y}})\|_2^2 + \lambda_1 \|(\phi(\mathbf{y}) - \phi(\hat{\mathbf{y}})\|_2^2 + \lambda_2 R_V(\hat{\mathbf{y}}) \tag{3.4}$$

$$\hat{\mathbf{y}} = \phi^{-1}(\phi(\mathbf{y})) \tag{3.5}$$

where $\hat{\mathbf{y}}$ is the reconstructed image from ϕ^{-1}, $\mathbf{y}$ is the ground-truth image from training data, and the multipliers, λ_1 and λ_2, are hyper-parameters to balance different terms. The first term in the loss function $\mathcal{L}_{inverted}$ is the image reconstruction error. In the second term, we use the distance between deep feature representations as the perceptual loss. The third term is the total variation regularizer to encourage smooth transitions between pixels.

With a well-trained inverted deep convolutional network, we can now map the final output image's feature representation $\phi(\mathbf{y}_\alpha)$ to the final output image $\mathbf{y}_\alpha$ in pixel space in milliseconds.

3.2 Implementation Details

This section dives into implementation details of our image editing approach. We will describe how to design and train the three models used in the approach.

3.2.1 Conditional Generative Adversarial Networks

We use the *pix2pix*[9] framework for the cGAN model. The generator in *pix2pix* uses the U-Net[17] architecture, which is an encoder-decoder with skip connec-

tions between the encoding layers and the corresponding decoding layers. And its discriminator, PatchGAN, classifies if an image is real or fake at the scale of patches. Specifically, we used 10 convolutional layers for U-Net, i.e., 5 layers for the encoder and 5 layers for the decoder. We used 5 convolutional layers to train a 70×70 PatchGAN, which classifies the input image at the scale of 70×70 patches.

Pix2pix requires paired data for training, but they are difficult to collect. For example, it's barely possible to take pictures of a person being young and old with exactly the same background settings and facial expressions. However, we solved this problem by generating such paired data through DFI since DFI itself doesn't require paired data and outputs images in good qualities. For the task of adding mustache, we used DFI to generate 500 images with mustache constrained with an intensity of 1. Then we paired the generated images with the input images to train a *pix2pix* model. Similarly, we generated 500 images with aging by DFI and trained another *pix2pix* model for the task of aging.

3.2.2 Deep Convolutional Networks

The DFI paper[19] has proved that convolutional layers of the normalized VGG-19 pertained on ILSVRC2012 are effective for linear interpolation. Therefore, we also choose the output vectors from the first layer of convolutional block 3, block 4, and block 5 in VGG-19, denoted respectively as *conv3_1*, *conv4_1*, and *conv5_1*, together as the deep feature representation. In our image editing approach, we only retain the parts of VGG-19 that are necessary to extract the deep feature representation, i.e., all layers before the second layer of convolutional block 5.

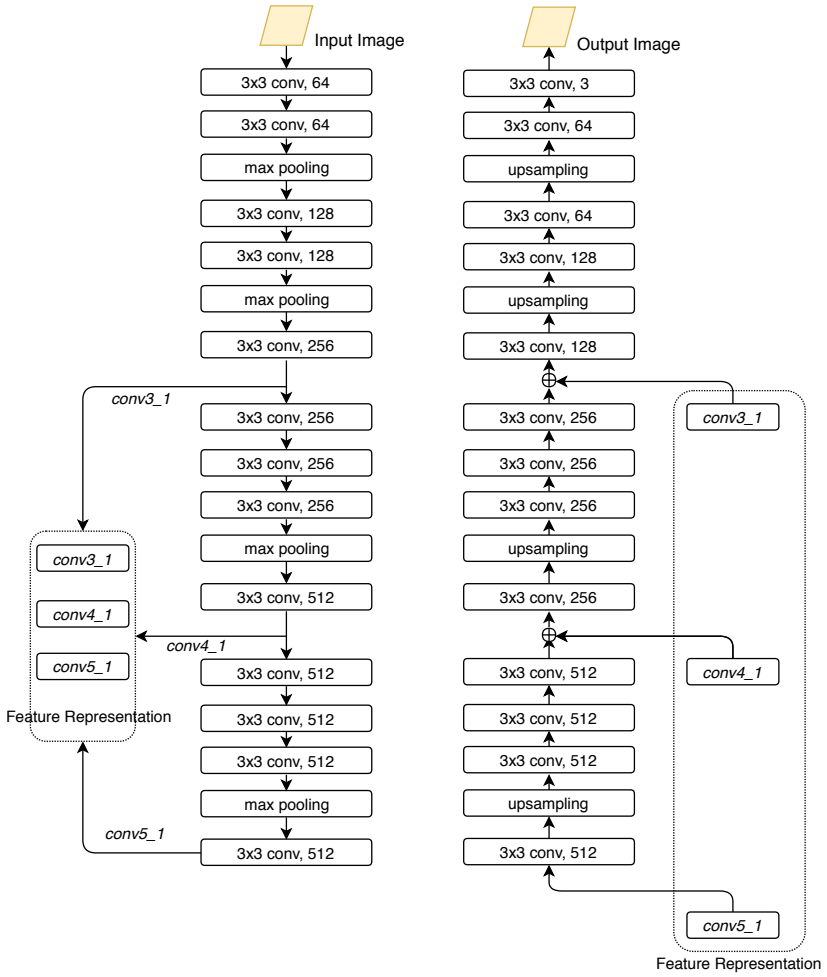


Figure 6: Left: The retaining parts of VGG-19. Right: The inverted network of VGG-19. VGG-19 outputs the deep feature representation, including three vectors: *conv3_1*, *conv4_1*, and *conv5_1*, which are inputted into the inverted network VGG-19.

The left figure in Figure 6 illustrates from which layers we extract the deep feature representation in VGG-19.

3.2.3 Inverted Deep Convolutional Networks

Similarly to [5], our inverted deep convolutional network is designed by inverting each layer in the retaining VGG-19. As illustrated in Figure 6, each con-

volutional layer in VGG-19 is replaced by a convolution layer by reversing the numbers of the input and output channels. Each max-pooling layer is replaced by an upsampling layer. Specifically, we use the 2D nearest neighbor upsampling. One thing different from [5] is that we have three vectors to input for each deep feature representation: *conv3_1*, *conv4_1*, and *conv5_1*. The first input *conv5_1* will be fed into the first layer of the inverted network as usual. *conv4_1* will be fed into the mirrored layer of the layer that outputs *conv4_1* in VGG-19, by concatenating with the feature map from the previous layer of this mirrored layer in the inverted network. The concatenation requires to double the input channel number for the convolutional layer, but we still keep the output channel number unchanged. *conv3_1* is fed into the network in the same way.

We train the inverted deep convolutional network by integrating the pre-trained VGG-19 into the training process as an decoder-encoder architecture. The inverted deep convolutional network learns to reconstruct an input image by minimizing the reconstruction error and the perceptual loss through VGG-19, as indicated in the loss function (Equation 3.4). We used CelebA dataset[14] for training. We optimized the loss function with Adam Optimizer[12] where lr = 0.001, $\beta_1 = 0.9$, and $\beta_1 = 0.999$, and the multipliers are $\lambda_1 = 1$ and $\lambda_2 = 0$.

The above described how to train an inverted deep convolutional network. We used the deep feature representation from VGG-19 as the input to train the inverted network. But in our image editing approach, the input for the inverted network is the output image's deep feature representation constructed via linear interpolation and the output image is the image edited with the desired attribute.

3.3 Evaluation

We introduce a new real-time image editing approach, which is time-efficient, memory-efficient, and user-controllable. To process a 512×512 image, our approach takes 0.055s on average with one NVIDIA TITAN X GPU. The generator for *pix2pix* occupies 64MB in memory. The retaining parts of the pre-trained VGG-19 occupy 50MB and its inverted network occupies 55MB. Therefore, the whole image editing approach occupies 169MB in memory. Most importantly, our image editing approach can generate images with a user-selected intensity of the attribute. Also, if we want to have more choices for the attribute to edit, our approach only requires to train a *pix2pix* model for the specific attribute, and the deep convolutional network and its inverted network can be shared across different attributes. Examples of our generated images are shown in Figure 7 and Figure 8. Figure 7 shows images having mustache in an increasing intensity from left to right. Figure 8 shows images of an increasing intensity of aging from left to right.

Table 1 compares the image editing performance among our approach, and *pix2pix*, and DFI. While *pix2pix* only takes about 0.02s to process an image and occupies 64MB in memory, it's not possible to adjust the intensity of the image editing. In turns, DFI allows users to control the intensity, but it takes roughly 5min to process an image and requires a face dataset aside during inference time. Compared to DFI, our approach decreases the inference time by 5000 times, and reduces the usage of memory space by 23 times if DFI uses CelebA dataset as the face dataset, which occupies 3.8GB after the face alignment preprocess as DFI requires. Figure 9 is a comparison of images generated by our

approach and DFI. As we can see, our approach is not only more time-efficient and more memory-efficient than DFI, but also outputs photo-realistic images with good qualities as DFI.

	Ours	DFI	*pix2pix*
Time(s)	0.055	300	0.02
Memory(MB)	169MB	3900MB	64MB
User-controllable	Yes	Yes	No

Table 1: Performance evaluation on processing an 512×512 image.

Figure 7: For each row, the first image is the input, and the second to the last images are generated using our image editing approach to add mustache with an increasing intensity α = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0 respectively.

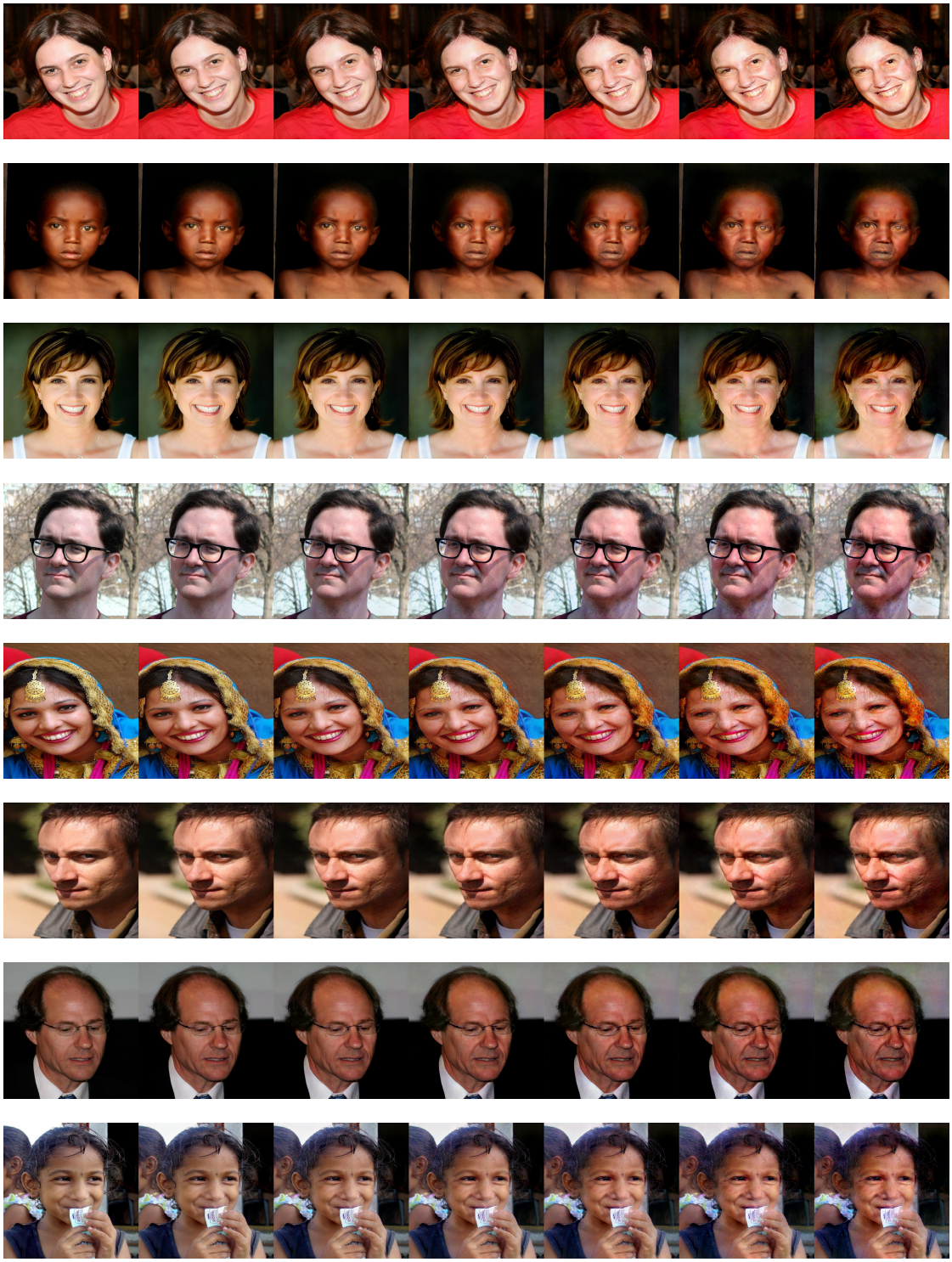


Figure 8: For each row, the first image is the input, and the second to the last images are generated using our image editing approach to age with an increasing intensity α = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0 respectively.

Figure 9: We compare images generated by our approach and DFI on the task of adding mustache. For each set of comparison, the first row is by our approach and the second row is by DFI. Images in the first column are the input images. Images from the second column to the last column are generated with an increasing intensity α = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0 respectively. Both our approach and DFI present photo-realistic results. Since our approach and DFI compose the mustache in different ways, the actual intensity of mustache presented by each approach may vary even with the same α.

CHAPTER 4

IMPLEMENTATION IN IOS APPLICATION

Due to the limited resource on mobile devices, power consumption and memory usage are major considerations for mobile applications. Under this circumstance, image editing algorithms like DFI can not be implemented on mobile devices since they may explode the memory. On the other hand, being time-efficient and memory-efficient makes it possible to implement our image editing approach on mobile devices. While cGANs are also time-efficient and memory-efficient, the "user-controllable" feature of our approach brings diversities and controls of the image editing to the application side, making the application interesting and pragmatic. In this chapter, we implement our image editing approach in an iOS application with two image editing functionalities, adding mustache and aging, to demonstrate that our approach is practical in real-world applications.

4.1 Implementation Details

We use Core ML[1] to integrate four pre-trained models into the iOS application: a *pix2pix* generator for adding mustache, a *pix2pix* generator for aging, a VGG-19, and an inverted VGG-19. While the iOS implementation is straightforward by following what we described in Chapter 3, there are two implementation details we want to address when it comes to a real-world application.

Face Detection: A real-world application should be prepared to deal with all

possibilities of the input image from users. Here we use the face detection to handle two user cases related to unexpected input images. In the first case, we use face detection to make sure that a face is presented in the input image before any further processing since our application is designed to only edit facial images. In the second case, due to that our *pix2pix* models are trained on headshots, it's difficult for them to process images where the face is small or not centered in the image layout, or there are multiple faces in the image. To avoid this, we perform face detection on every input image, and crop out a headshot for further processing. The processed headshot will be pasted back to its original position in the input image to composite the final output image. Figure 10 gives an example where the face is small in the input image, and demonstrates how an image flows in this process. In practice, we used the built-in face detection algorithm on iPhone.

Color Correction: It's common that generative models output images with color distortion. But when we composite the output image by directly pasting a processed headshot back to the input image, the color distortion will result in a floating box over the output image (Figure 10(d)). We resolve this artifact by applying alpha blending. The idea of alpha blending is to make a foreground image translucent so that the foreground image can be composited with a background image seamlessly. In our case, we make boundaries of the processed headshot gradually translucent from outside inwards but keep the face opaque such that the image editing is not affected (Figure 11). Then the headshot can be composited with the input image seamlessly. Figure 12 shows an output image without the artifact.

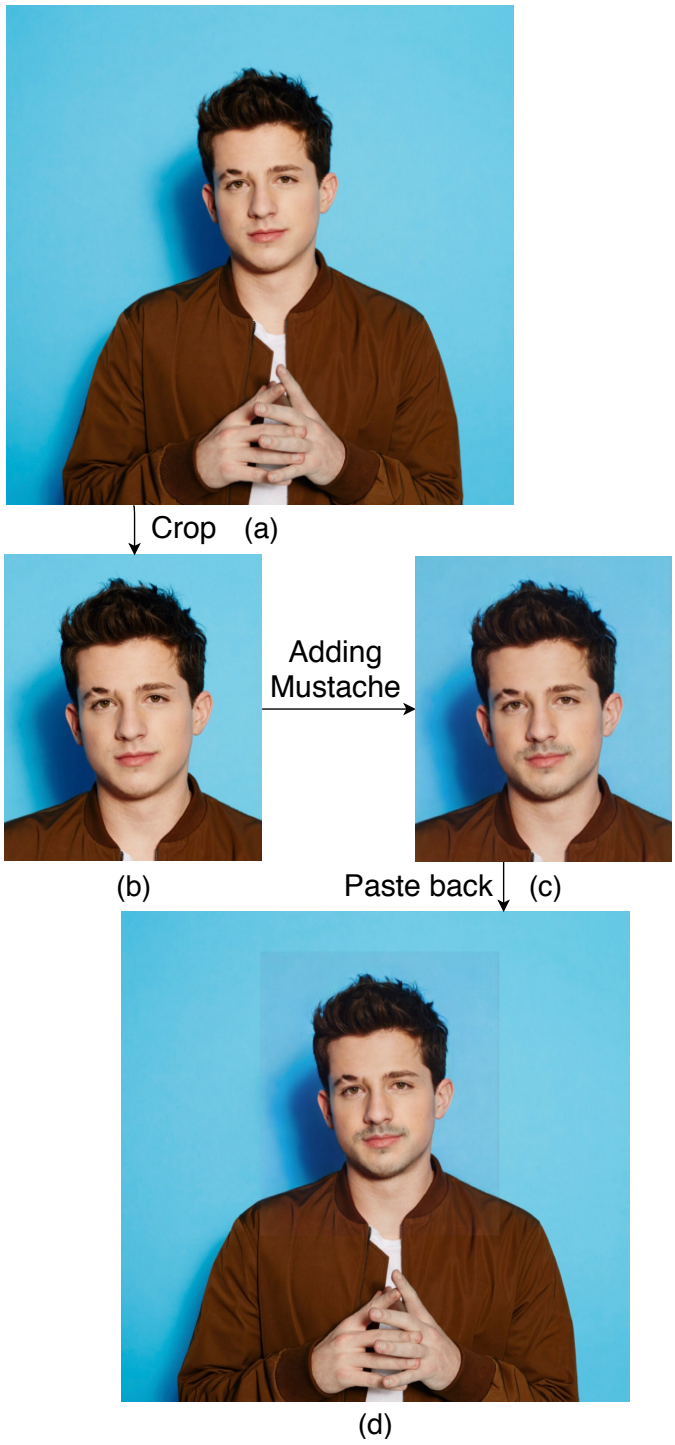
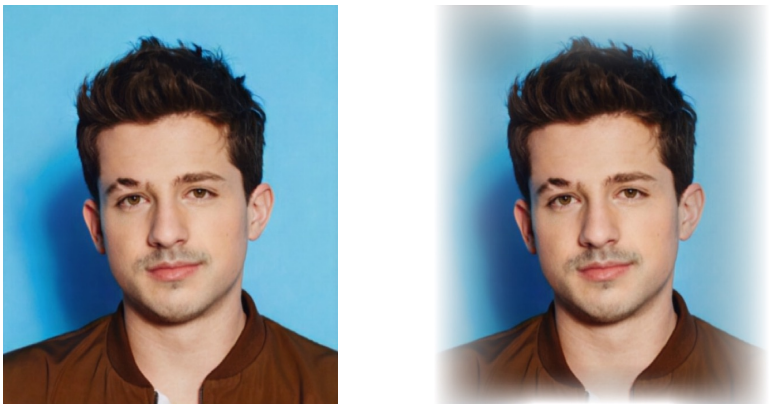


Figure 10: Image(a) is the input image. Image(b) is the headshot cropped from the input image with face detection. Image(c) is the processed headshot with our image editing approach to add mustache. Image(d) is the final output image if we directly paste the processed headshot to the input image (Image(a)). There is an obvious floating box over the face on Image(d).



(a) (b)

Figure 11: Image(a) is the processed headshot (same as Figure 10(c)). Image(b) is the processed headshot with gradually translucent boundaries from outside inwards.

Figure 12: This image the final output image in the iOS application, obtained with alpha blending by compositing the processed headshot (Figure 11(b)) with the input image (Figure 10(a)). There is no floating box on this final output image.

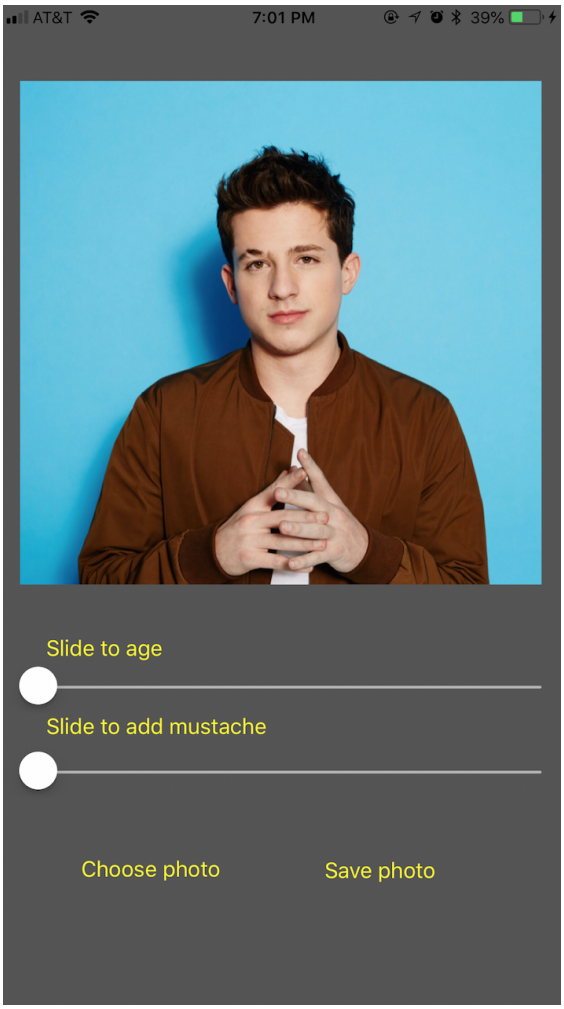


Figure 13: This is a screenshot from the application. The application can edit facial images to add mustache or age. The sliders are used to control the intensity of the image editing.

4.2 Functionalities

This iOS application performs facial image editing with two attributes: adding mustache and aging. Users can input an image either from the photo gallery or the camera. If there is no face in the image, the application will prompt that

"No face detected". If the image is valid for the image editing, users can move sliders to control how much mustache or aging to be applied on their images. Finally, users can save the edited image in the photo gallery.

4.3 Performance

We installed the iOS application on a 128GB iPhone 7 Plus. The GPU on iPhone 7 Plus uses a custom version of the PowerVR GT7600 GPU, and shares the 128GB system memory with the CPU. This iOS application occupies roughly 300MB in memory with the four models. In practice, it first takes about 5s to preprocess the image, including cropping out the headshot with face detection (1s) and generating two intermediate images, one for adding mustache (2s) and one for aging (2s). Then, it takes another 2s to generate the final output image with a specified intensity control factor on a desired attribute. Linear interpolation and reverse mapping are performed during this 2s. Overall, this iOS application takes 5s for an end-to-end image editing process of a single attribute. Compared to the performance on TITAN X (i.e., 0.055s), our image editing approach runs about 90 times slower on the iPhone. We explain this by that the iOS application involves other processes, such as face detection and alpha blending. Also, iPhone can not fully utilize the GPU power for this iOS application since there are always other processes ongoing on iPhone. Nevertheless, this application is still time-efficient, memory-efficient, and user-controllable.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we presented a real-time image editing approach and implemented the approach in an iOS application to demonstrate its practicability in real-world applications. We showed how to effectively combine strengths of two image editing algorithms, conditional Generative Adversarial Networks and Deep Feature Interpolation, to develop an image editing approach that is not only time-efficient and memory-efficient, but also allows users to control the intensity of the image editing. We also illustrated how to design and train an inverted deep convolutional network to map deep feature representations into images in pixel space. With the implementation of the iOS application, we discussed how to handle unexpected input images and how to alleviate the color distortion artifact.

In the future, we will work on further reducing the memory usage for our image editing approach in two directions. The first direction is to study cGANs for multi-domain image editing so that we don't need to store one *pix2pix* model for each attribute. One related work is StarGAN[4]. In the second direction, we will investigate and integrate light-weighted deep convolutional networks[20][8][7] in our image editing approach.

BIBLIOGRAPHY

[1] *Core ML*. `https://developer.apple.com/documentation/coreml`.

[2] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *International Conference on Machine Learning*, pages 552–560, 2013.

[3] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[4] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. *arXiv preprint arXiv:1711.09020*, 2017.

[5] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4829–4837, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[8] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*, 2017.

[9] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[10] Ira Kemelmacher-Shlizerman, Supasorn Suwajanakorn, and Steven M Seitz. Illumination-aware age progression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3334–3341, 2014.

[11] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jungkwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*, 2017.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. 2015.

[16] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[18] Zhixin Shu, Eli Shechtman, Dimitris Samaras, and Sunil Hadap. Eyeopener: Editing eyes in the wild. *ACM Transactions on Graphics (TOG)*, 36(1):1, 2017.

[19] Paul Upchurch, Jacob Gardner, Kavita Bala, Robert Pless, Noah Snavely, and Kilian Q Weinberger. Deep feature interpolation for image content changes. *arXiv preprint arXiv:1611.05507*, 2016.

[20] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[21] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.